Estudo empírico: detecção de *Code Smells* com aprendizado de máquinas

Raimundo Alan Freire Moreira[1], **Lucas José Lemos Braz**[1], **Fischer Jônatas Ferreira**[2], **Márcio André Baima Amora**[1]

[1] Universidade Federal do Ceará (UFC) – Sobral – CE – Brasil

[2]Universidade Federal de Itajubá – Itabira – MG – Brasil

alan.freire@alu.ufc.br, lucas.jlb19@alu.ufc.br

fischer.ferreira@sobral.ufc.br, marcio@sobral.ufc.br

Abstract. *Detecting code smells during the software development process is important for enhancing software quality, and refactoring is essential for eliminating these indicators of problems. This study evaluates an empirical approach based on training five machine learning algorithms to detect code smells in software systems, using software metrics as parameters. The results demonstrate that the machine learning approach performed exceptionally well in detecting code smells, achieving an accuracy ranging from 93.7% to 99.2%.*

Resumo. *A detecção de code smells durante o processo de desenvolvimento de software é importante para melhorar a qualidade do software e a refatoração é fundamental para eliminar esses indícios de problema. Este estudo avalia uma abordagem empírica que se baseia no treinamento de cinco algoritmos de aprendizado de máquina para detectar code smells em sistemas de software, utilizando métricas de software como parâmetros. Os resultados mostram que a abordagem de aprendizado de máquina tive um excelente desempenho para a detecção de code smells, alcançando uma acurácia entre 93,7% a 99,2%.*

1. Introdução

Code smell é uma característica estrutural do software que indica um aspecto no código ou no design que pode causar problemas na manutenção do software. *Code smell* não é um erro no sistema, pois não impede o funcionamento do programa, mas pode aumentar o risco de falha do software ou desacelerar o desempenho. Notavelmente, a predição precoce do *code smell* durante a fase de desenvolvimento é muito importante, especialmente em projetos de código-fonte em grande escala [Abdou and Ramadan, 2022]. O processo de refatoração é fundamental para eliminar os *code smells* e melhorar a manutenção e a qualidade do software. Fowler [2018] apresenta uma definição de 22 tipos de *code smells* no código-fonte e oferecem algumas operações de refatoração para corrigi-los.

O impacto dos *code smells* no software foram examinados por vários estudos e revelaram seu efeito indesejável na qualidade do software [Yamashita and Moonen, 2012, 2013; Yamashita and Counsell, 2013; Sahin et al., 2014]. Eles também investigaram os resultados da remoção dos *code smells* na redução da probabilidade de falhas e erros no sistema de software. Analisaram os desafios decorrentes dos *code smells*, os quais

têm efeitos adversos no processo de desenvolvimento de software, e recomendaram a aplicação de refatoração no software para eliminá-los.

Dewangan et al. [2021] falam que na detecção de *code smells*, as métricas desempenham um papel fundamental ao medir tanto as qualidades funcionais quanto não funcionais do software, além de contribuir para a compreensão dos recursos do código-fonte. As informações estáticas do software são obtidas por meio de métricas como o número de métodos, classes e parâmetros, além da avaliação do acoplamento e coesão entre os objetos do sistema.

Trabalhos anteriores investigam a detecção de *Code Smell* por meio de detecção manual [Travassos et al., 1999; Ciupke, 1999], Moha et al. [2009] e Tsantalis and Chatzigeorgiou [2009] introduziram abordagens baseados em métricas, Yamashita and Moonen [2013] e Tarwani and Chug [2016] avaliam problemas de manutenabilidade causado por *Code Smell* e Fontana et al. [2016] e Dewangan et al. [2021] abordaram a detecção por aprendizado de máquina. Porém, o tema ainda está em aberto quando a redução de custo para a atividade de detecção de *Code smell* poder ser incorporado no processo de desenvolvimento de software. Assim, com o uso de detecção automatizada por meio de ML pode ser uma alternativa viável para que a detecção de *Smell* ocorra em todas as etapas do desenvolvimento.

Neste artigo é buscado responder sobre a eficácia dos algoritmos de aprendizado de máquina na detecção de *code smells* e se a técnica de validação cruzada melhora os resultados e ameniza as ameaças a validade dos resultados dos algoritmos de aprendizado de máquina neste caso de uso. Para isso foi adotado o conjunto de dados de *code smells* contendo quatro tipos: *God Class*, *Data Class*, *Feature Envy* e *Long Method*. Cinco algoritmos de aprendizado de máquina (*Multilayer Perceptron* - MLP, Árvore de Decisão, Floresta Aleatória, Gradiente Boost e Máquina de Vetores de Suporte - SVM) são aplicados nos conjuntos de dados e alcançaram o melhor desempenho(acurácia) no algoritmo de Floresta Aleatória (99,2%).

A contribuição buscada neste trabalho é avaliar os cinco métodos de aprendizado de máquina para a detecção de *code smells*. Em seguida, são apresentadas as métricas de desempenho (acurácia, precisão, sensibilidade e F1 score) obtidas pelo treinamento dos métodos de aprendizado de máquina para comparação dos mesmos, fazendo também a comparação dos resultados usando a técnica de validação cruzada e sem a técnica de validação cruzada.

Este trabalho está organizada da seguinte forma: Na Seção 2 faz um descritivo sobre *Code Smells*, Métricas de Software e Aprendizado de Máquinas com uma breve descrição de cada modelo usado. Também é feito um descritivo sobre Normalização de Dados e Validação Cruzada. Já a metodologia usada para o desenvolvimento deste trabalho está localizada na Seção 3. Os resultados e analise dos dados na Seção 4, a Seção 5 cita alguns trabalhos relacionados, na Seção 6 é discutido as ameaças a pesquisa e a conclusão é discutida na Seção 7.

2. Fundamentação teórica

2.1. *Code Smells e Métricas de Software*

A presença de *code smells* resulta em desafios significativos, incluindo complicações na manutenção, aumento da complexidade e redução da legibilidade do código. A manutenção de software em grande escala é particularmente problemática devido ao tamanho e à complexidade envolvidos, sendo mais de 80% do custo total do software dedicado à manutenção. Cerca de 60% do tempo durante a manutenção é gasto na compreensão do código. Portanto, a existência dos *code smells* no código agrava essas dificuldades, tornando a manutenção do software ainda mais complexa [Abdou and Ramadan, 2022].

Para este artigo, foi feito as comparações de resultados dos algoritmos de aprendizado de máquina com os seguintes *code smells* conjunto de dados de Abdou and Ramadan [2022]: ***God Class*** descreve uma classe extensa que possui muitas linhas de código, funções ou campos [Riel, 1996; Brown et al., 1998; Fontana et al., 2016; Dewangan et al., 2021]; ***Data Class*** aponta para a classe usada para armazenar os dados utilizados por outras classes. *Data Class* abrange apenas campos e métodos de acesso (*getters/setters*) [Riel, 1996; Fontana et al., 2016; Dewangan et al., 2021]; ***Feature Envy*** é um *code smell* ao nível de método que acessa dados ou utiliza operações pertencentes a classes diferentes da sua própria classe, ou seja, ele utiliza dados estrangeiros adicionais em comparação com os dados locais [Fontana et al., 2016; Dewangan et al., 2021]; ***Long Method*** é um *code smell* ao nível de método que se refere a um método de grande tamanho, devido ao número de linhas de código e funcionalidades presentes no método [McConnell, 2004; Fontana et al., 2016; Dewangan et al., 2021].

Para avaliar a qualidade do código, as métricas de software desempenham um papel crucial na engenharia de software. As métricas cobrem aspectos como coesão, acoplamento, complexidade e tamanho nos níveis de método, classe, projeto e pacote. As métricas ao nível de método incluem complexidade ciclomática, tamanho do método e contagem de parâmetros. No nível de classe abrangem coesão, acoplamento, tamanho da classe e contagem de métodos. Em nivel de projeto incluem complexidade, tamanho e contagem de classes, enquanto as métricas ao nível de pacote envolvem coesão, acoplamento, complexidade e tamanho [Ashraf Abdou, 2022].

2.2. Aprendizado de máquina

No Aprendizado de Máquina, segundo Ray [2019], um programa de computador é designado para realizar algumas tarefas e diz-se que a máquina aprendeu com sua experiência se seu desempenho mensurável nessas tarefas melhora à medida que adquire cada vez mais experiência na execução delas. Portanto, a máquina toma decisões e faz previsões/-baseadas em dados.

Neste trabalho foi escolhido cinco algoritmos de aprendizado de máquinas para fazermos a comparação no desempenho de cada um quanto a detecção de *code smells*, são eles: ***Multilayer Perceptron*** ou **MLP** [Ruck et al., 1990], **Árvore de Decisão** [Quinlan, 1990], **Floresta Aleatória** [Breiman, 2001], **Gradiente Boost** [Friedman, 2001] e ***Support Vector Machines*** ou **SVM** [Noble, 2006].

2.3. Normalização e validação dos dados

A normalização é uma técnica crucial para harmonizar escalas de conjuntos de dados numéricos [Al Shalabi et al., 2006]. Essa etapa é essencial para garantir o equilíbrio em modelos, especialmente aqueles baseados em diferentes métricas ou otimizações baseadas em gradiente.

Ao aplicar esta técnica, um vetor de atributos x em x' usando um escalonamento (E') e fatores de tradução (T): $x' = Ex + T$. Métodos como escalonamento para um intervalo, dimensionamento logaritmo, Z-score e Min-Max atendem a diversas distribuições de dados e requisitos de modelo [Han et al., 2022]. Neste trabalho, utilizamos a normalização Min-Max, que dimensiona os valores para $[0, 1]$, dada por: $x' = \frac{(x-min(x))}{(max(x)-min(x))}$ [Han et al., 2022].

Por outro lado, a validação é crucial para evitar *overfitting*, garantindo que o modelo generaliza bem para dados inéditos [Arlot and Celisse, 2010]. Matematicamente, dado o conjunto de dados D com N amostras (X, y), onde X representa a matriz de atributos de entrada e y o vetor alvo, treinamos o modelo M em um subconjunto D_{train} e avaliar seu desempenho em um subconjunto de validação D_{val}, onde $D = D_{train} \cup D_{val}$.

3. Metodologia

Questões problemas abordados no artigo:

1. Qual a eficácia das técnicas de *machine learning* MLP, Árvore de Decisão, Floresta Aleatória, Gradiente Boost e SVM para detecção de *code smell*?
2. Qual o impacto da validação cruzada no conjunto de dados utilizado?

Neste estudo foi definido como objetivo principal a avaliação das técnicas de aprendizado de máquina para a detecção de *code smells* e como objetivo específico avaliar o impacto da validação cruzada na precisão dos algoritmos de aprendizado de máquina.

As etapas seguidas por esse trabalho são mostradas na Figura 1. Inicialmente o conjunto de dados é coletado, em seguida é feito a etapa de pré-processamento (Normalização) é aplicada ao conjunto de dados para cobrir as diferentes faixas dos conjuntos de dados e obter os melhores parâmetros dos algoritmos. Em seguida, os algoritmos de aprendizado de máquina no conjunto de dados são treinados e seu desempenho é calculado. A técnica de validação cruzada em 10 partes é aplicada para avaliar e comparar o desempenho de cada experimento durante o processo de treinamento.

3.1. Conjunto de dados

Os conjuntos de dados usados neste estudo são fornecidos por Abdou and Ramadan [2022] que se basearam no trabalho de Fontana et al. [2016], definidos como dados de referência. Este conjunto de dados pertence a diferentes domínios, sendo compostos por 76 sistemas de diferentes tamanhos escritos em Java. Em Fontana et al. [2016] é descrito o conjunto de métricas orientadas a objetos aplicados em 76 sistemas do Qualitas Corpus (QC) de Tempero et al. [2010], abrangendo os níveis de método, classe, pacote e projeto. Algumas dessas métricas foram definidas conforme o aspecto da qualidade de software, como complexidade, tamanho e acoplamento. Outras métricas dependem da contagem da composição em pacotes ou classes. Os *code smells* foram definidos no nível de método

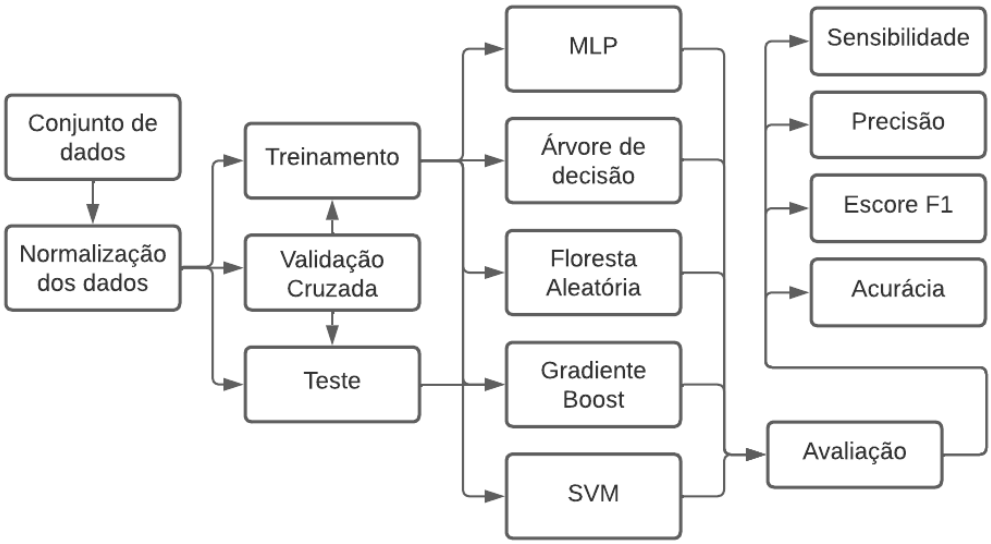

Figura 1. Fluxo de trabalho proposto.

ou classe. No nível de método, foi escolhido para detectar *Feature Envy* e *Long Method*, enquanto no nível de classe, detectamos *Data Class* e *God Class*. Os autores do conjunto de dados escolheram estes quatro *code smells* devido sua alta incidência e por ter grande impacto negativo na qualidade dos softwares.

A Tabela 1(a) mostra o tamanho geral dos sistemas selecionados. Abdou and Ramadan [2022] diz que o grande número de sistemas heterogêneos usados neste estudo é fundamental para garantir que os resultados do processo de aprendizado de máquina não dependam de um conjunto de dados específico e para permitir a generalização dos resultados obtidos.

As ferramentas empregadas para identificar os *code smell* são apresentadas na Tabela 1(b). Essas ferramentas foram selecionadas baseadas nas seguintes regras: serem gratuitas, de fácil configuração e generalização e terem regras de detecção diferentes das demais.

(a) **Características dos sistemas. Fonte: [Fontana et al., 2016]**

Característica	Valores
Quantidade de Sistemas	74
Quantidade de Pacotes	3.420
Quantidade de Classes	51.826
Quantidade de Métodos	404.316
Linhas de Código	6.785.568

(b) **Detectores de *code smells*. Fonte: [Abdou and Ramadan, 2022]**

Code Smell	Ferramenta: regras de detecção
Long method	iPlasma, PMD
Data class	iPlasma, Fluid Tool, Anti-Pattern Scanner
God class	iPlasma, PMD
Feature Envy	iPlasma, Fluid Tool

Tabela 1. Características e ferramentas

O conjunto de dados está dividido como mostrado na Tabela 2, onde os mesmos são classificados como falsos positivos e verdadeiros positivos. Essa classificação se da através da análise feita por [Abdou and Ramadan, 2022], onde o mesmo cria uma tabela de severidade dos *code smells* fazendo uma divisão em quatro níveis, onde o nível 1 indica não haver presença de *code smell*, assim classificamos como falsos positivos. Os demais

níveis classificamos verdadeiros positivos, por haver presença de *code smells* no projeto.

Tabela 2. Composição do conjunto de dados.

Code Smell	Falsos Positivos	Verdadeiros Positivos	Total
Long method	280	140	420
Data class	151	269	420
God class	154	266	420
Feature Envy	280	140	420

As métricas de software usadas estão descritas no trabalho de [Abdou and Ramadan, 2022], essas foram selecionadas e classificadas para abranger diversos aspectos do controle de qualidade de software, como Acoplamento, Complexidade, Coesão, Tamanho, Herança e Encapsulamento.

3.2. Avaliação de desempenho

Para medir o desempenho dos algoritmos de aprendizado de máquina neste trabalho, foram considerados quatro parâmetros de desempenho: precisão, sensibilidade, pontuação F1 e Acurácia. Para calculá-los, são utilizados verdadeiro positivo (TP), verdadeiro negativo (TN), falso positivo (FP) e falso negativo (FN).

Esses parâmetros são calculados usando uma matriz de confusão que contém as informações reais e previstas reconhecidas pelos classificadores de detecção de padrões de projeto [Catal, 2012]. As equações para o cálculo das avaliações de desempenho são mostradas na Tabela 3.

Tabela 3. Fórmulas das Métricas de Avaliação. Fonte: [Hossin and Sulaiman, 2015]

Métricas	Formula
Acurácia (Accuracy)	$\frac{VP+VN}{VP+FP+VN+FN}$
Precisão (Precision)	$\frac{VP}{VP+FP}$
Sensibilidade (Recall)	$\frac{VP}{VP+FN}$
F1-Score	$2 \times \frac{\text{Precisao} \times \text{Recall}}{\text{Precisao} + \text{Recall}}$

4. Resultados

Neste artigo, como mencionado anteriormente, foi adotado cinco técnicas de aprendizado de máquina para predição de *code smells* em projetos de software. Foram analisados quatro tipos de *code smells* divididos em dois níveis: *smells* de classe e *smells* de métodos.

Na Tabela 4 é mostrado uma comparação entre os resultados das métricas com todos os *code smells* usando a técnica *holdout* dividindo o conjunto de dados em 70% para treino e 30% para validação. Como é possível observar os algoritmos tem um ótimo desempenho. Com resultados variando entre 89,7% para o algoritmo Árvore de Decisão no *code smell Data class* em seu pior desempenho e 99,2% para os algoritmos Gradiente *Boost*, Floresta Aleatória e Árvore de Decisão treinados para o *code smell Long Method* no melhor desempenho.

Tabela 4. Comparação das métricas

Code Smells	Algoritmo	Precisão	Sensibilidade	Pontuação F1	Acurácia
Data Class	MLP	93,1%	93,7%	93,4%	93,7%
	Árvore de Decisão	88,9%	90,0%	89,3%	89,7%
	Floresta Aleatória	96,7%	**96,7%**	**96,7%**	**96,8%**
	Gradiente Boost	**97,0%**	96,3%	96,6%	**96,8%**
	SVM	91,3%	91,2%	91,3%	92,3%
God Class	MLP	**92,9%**	91,7%	92,3%	92,9%
	Árvore de Decisão	88,9%	89,2%	89,0%	89,7%
	Floresta Aleatória	84,1%	**92,4%**	**93,1%**	**93,7%**
	Gradiente Boost	**92,9%**	91,7%	92,3%	92,9%
	SVM	91,1%	87,0%	88,5%	89,7%
Feature Envy	MLP	93,2%	93,8%	93,5%	94,4%
	Árvore de Decisão	95,7%	96,6%	96,3%	96,8%
	Floresta Aleatória	**98,1%**	**98,1%**	**98,1%**	**98,4%**
	Gradiente Boost	97,5%	96,8%	97,2%	97,6%
	SVM	91,2%	89,8%	89,2%	91,3%
Long Method	MLP	95,2%	95,8%	95,5%	96,0%
	Árvore de Decisão	**99,4%**	98,8%	**99,1%**	**99,2%**
	Floresta Aleatória	98,8%	**99,4%**	**99,1%**	**99,2%**
	Gradiente Boost	**99,4%**	98,8%	**99,1%**	**99,2%**
	SVM	92,8%	92,8%	92,8%	93,7%

Analisando os dados gerado pelo algoritmo MLP, é visto que seu melhor desempenho foi para o *code smell Data Class*, porém o mesmo ainda se mostra muito eficiente na detecção dos demais *code smells*. Para o algoritmo da Árvore de decisão, o seu melhor resultado foi para o *code smell Long Method*, alcançou um excelente valor para a acurácia de 99,2%, mas também teve o pior desempenho entre os algoritmos analisados para os *code smells Data Class* e *God Class*. O SVM se mostrou muito promissor, como sua melhor pontuação para *Long Method* (93,7%) e para *Data Class* e *Feature Envy* alcançou valores de acurácia acima de 91%. Os dados para Floresta Aleatória são os mais promissores, pois a mesma atingiu as melhores pontuações nos quatro code smells, com destaque para Long Method que o valor ficou acima de 99%. Já o Gradiente *Boost* ficou logo atrás da Floresta Aleatória com pontuação igual nos *code smells Data Class* e *Long Method*, mas um pouco menos eficiente para *God Class* e *Feature Envy*.

Tabela 5. Desempenho dos algoritmos com validação cruzada em 10 partes (Acurácia)

	Data Class	God Class	Featury Envy	Long Method
MLP	91,9%	91,6%	90,5%	98,3%
Árvore de Decisão	90,8%	91,2%	93,6%	98,6%
Floresta Aleatória	**94,2%**	**96,3%**	**94,5%**	**99,0%**
Gradiente Boost	93,2%	94,6%	93,9%	98,6%
SVM	91,2%	91,2%	88,5%	97,3%

Na Tabela 5 são apresentados os dados referentes ao desempenho dos algoritmos empregando a técnica de validação cruzada de 10 partes. E a seguir fazemos uma

comparação de desempenho entre as técnicas *holdout* e validação cruzada, porém essa comparação é apenas para efeito didático, já que a finalidade da validação cruzada é a mitigação de possíveis falhas no processo de treinamento do da técnica *holdout*.

Ao analisar os resultados dos algoritmos, é notório que, para os *code smells Data Class* e *God Class*, tanto os algoritmos MLP quanto Árvore de Decisão não demonstram melhora de desempenho. Por outro lado, os algoritmos Floresta Aleatória, Gradiente *Boost* e SVM mostraram melhorias no reconhecimento de *God Class*. No caso do *code smell Feature Envy*, os resultados foram contrários às expectativas, resultando em uma redução média de eficiência de 3,5%. Finalmente, em relação ao *code smell Long Method*, somente os algoritmos MLP e SVM foram beneficiados positivamente pela validação cruzada.

Podemos ver que os resultados da técnica de validação cruzada (Tabela 5) está bem parecida com a técnica *holdout* (Tabela 4), evidenciando que possíveis vícios, *overfit* ou outros problemas relacionados ao treinamento com o conjunto de dados adotado estão mitigados.

4.1. Análise dos resultados

Para responder a primeira questão levantada neste trabalho, na qual é indagado sobre a eficiência das cinco técnicas de aprendizado de máquina na detecção de *code smells*, é exposto na Tabela 4 que as mesmas alcançam valores superiores a 93% para o algoritmo Floresta Aleatória, demonstrando ser um caminho promissor para desenvolvimento de ferramentas para auxiliar programadores a escrever códigos de melhor qualidade, redução da complexidade, facilidade na manutenção e aumento da produtividade.

Quanto a resposta da segunda questão, que levanta uma interrogação sobre o impacto da validação cruzada em 10 partes, é possível ver os valores da Tabela 5 que mostra o desempenho dos algoritmos usando a técnica e fazer uma comparação com os números para acurácia da Tabela 4 que usa *holdout*, fica evidenciado que a validação cruzada mostra ganhos para os *code smells Feature Envy* e *Long Method*. Sendo que para o *Feature Envy* o algoritmo MLP não apresentou melhora no desempenho. E para *Long Method* a validação cruzada mostrou melhora no desempenho apenas em dois algoritmos: MLP e SVM.

5. Trabalhos Relacionados

Foram propostas diversas ferramentas para detecção de *code smells*, abrangendo tanto ferramentas comerciais quanto protótipos de pesquisa. Fontana and Zanoni [2017] falam que essas ferramentas utilizam uma variedade de técnicas para identificar *code smells*: algumas se baseiam em métricas [Lanza and Marinescu, 2007; Fontana et al., 2015], outras empregam uma linguagem de especificação própria [Moha et al., 2009], realizam análise de programas para encontrar oportunidades de refatoração [Tsantalis and Chatzigeorgiou, 2009, 2011], exploram a análise de repositórios de software [Palomba et al., 2015] ou ainda recorrem a técnicas de aprendizado de máquina. Focalizamos em abordagens que empregam técnicas de aprendizado de máquina.

Pushpalatha and Mrunalini [2021] propuseram uma abordagem de aprendizado de máquina para detectar *code smells* de software e observaram as métricas que desempenham papéis críticos no processo de detecção. Eles aplicaram algoritmo genético baseado

em duas técnicas de seleção de recursos e técnica de otimização de parâmetros baseada em uma pesquisa em grade. Eles obtiveram melhor precisão na previsão dos *code smells* de *Data Class*, *God Class* e *Long Method* em 98,05%, 97,56% e 94,31%, respectivamente, usando o método GA_CFS, e no *Long Method* obtiveram a melhor precisão de 98,38% usando o recurso GA-Naïve Bayes método de seleção.

Abdou and Ramadan [2022] apresentam um estudo comparativo de técnicas de aprendizado de máquina para classificar a gravidade de *code smells* em software. Os pesquisadores propuseram um modelo baseado em métricas de software e aprendizado de máquina para detectar esses problemas de design. Eles utilizaram diferentes abordagens, incluindo classificação multinomial, ordinal e regressão, e avaliaram a precisão na ordenação e classificação da gravidade dos *code smells*. Além disso, identificaram limitações e propuseram direções para pesquisas futuras, como a inclusão de mais tipos de *code smells* e a avaliação de técnicas de balanceamento de dados e aprendizado em conjunto.

Kaur and Kaur [2021] apresentaram a técnica de aprendizado de conjunto (*Ensemble Learning*) e a técnica de seleção de características de correlação em três conjuntos de dados Java de código aberto para detecção de *code smells*. Eles aplicaram o classificador *Bagging* e Floresta Aleatória para analisar cada abordagem com quatro medidas de desempenho: acurácia (P1), G-*mean* 1 (P2), G-*mean* 2 (P3) e F-*measure* (P4).

A Tabela 6 representa uma breve comparação dos resultados obtidos por nós com outros trabalhos relacionados. Esses métodos aplicaram técnicas de aprendizado de máquina em conjunto de dados de *code smells* fornecido por [Fontana et al., 2016].

Tabela 6. Comparação dos resultados (Acurácia).

Autor	Conjunto de Dados			
	Data Class	**God Class**	**Feature Envy**	**Long Method**
[Fontana and Zanoni, 2017]	77,0%	74,0%	93,0%	92,0%
[Di Nucci et al., 2018]	83,0%	83,0%	84,0%	82,0%
[Mhawish and Gupta, 2020]	99,7%	98,5%	98,0%	96,0%
[Abdou and Ramadan, 2022]	93,0%	92,0%	97,0%	97,0%
Presente trabalho	96,8%	96,3%	98,4%	99,2%

6. Ameaças a pesquisa

Neste estudo podemos considerar algumas ameaças à validade da pesquisa, primeiramente a validade interna, têm efeito na correção dos resultados do experimento, e o segundo tipo de validade é a validade externa, que está relacionada à capacidade de generalização desse estudo em outros contextos.

A validade interna está relacionada ao grau de erro nos resultados devido à avaliação manual dos *code smells*, juntamente com a capacidade dos desenvolvedores de compreenderem os problemas de definição dos *code smells* e o nível de experiência e conhecimento dos desenvolvedores em programação orientada a objetos. Esses fatores afetam a faixa de valores apropriados que devem ser atribuídos à variável de classe pelo avaliador de *code smells*, podendo causar distorção no conjunto de dados.

Em segundo lugar, a validade externa está relacionada à generalização dos resultados deste estudo, limitada por alguns fatores, como as fontes de dados, construídas a

partir de 76 sistemas pertencentes a diferentes domínios escritos em Java, o que não pode ser generalizado para software não-Java. Além disso, o grau de equilíbrio entre a ausência e a presença de *code smells* em conjuntos de dados para diferentes níveis de gravidade é um dos fatores que afetam a validade externa. Neste estudo, o grau de equilíbrio dos *code smells* é mantido na faixa de 33% a 66% e não é homogêneo nos conjuntos de dados.

7. Conclusão

Neste estudo propusemos uma abordagem baseada em aprendizado de máquina usando cinco algoritmos: MLP, Árvore de Decisão, Floresta Aleatória, Gradiente Boost e SVM e métricas de software para detectar *code smells* em sistemas de software. O conjunto de dados usado neste estudo foi retirado de [Fontana et al., 2016] para quatro tipos de *code smells*, conforme descrito na Subseção 3.1.

Duas são as principais contribuições desta pesquisa: na primeira, algoritmos de aprendizado de máquina são usados para identificar os quatro *code smells*. Na segunda etapa, as medidas de desempenho (Acurácia, Precisão, Sensibilidade e F1-score) são calculadas e usadas para avaliação do desempenho dos algoritmos. Também foi usada a validação em 10 partes para buscar a melhoria do desempenho das algoritmos usadas.

Observamos que as abordagens de aprendizado de máquina, Árvore de Decisão, Floresta Aleatória e Gradiente Boost tiveram um excelente desempenho para a detecção de *code smell Long Method* alcançando uma acurácia de mais de 99%, para *Feature Envy* valores superiores a 96% de acurácia. Já para *Data Class* o desempenho caiu um pouco para o algoritmo Árvore de Decisão ficando em 89,7%, os demais ficam em 96,8% de acurácia. E para *code smell God Class* a abordagem com Árvore de Decisão ganhou desempenho com a técnica de validação cruzada em 10 partes atingindo percentual de 91,2%, Floresta Aleatória conseguiu 96,3% e Gradiente Boost 94,6%. As técnicas de aprendizado de máquina MLP e SVM também tiveram um ótimo desempenho, mas abaixo das demais. Conseguindo valores de acurácia de 98,3% e 97,3% para *Long Method*, 94,4% e 91,3% para *Feature Envy*, 93,7% e 92,3% para *Data Class* e 92,9% e 91,2% para *God Class*.

Nos trabalhos futuros, pretende-se estender os experimentos para novos conjuntos de dados, incluindo mais tipos de *code smells*, buscar também formas de definições mais sólidas, ou melhor, generalizadas, para identificação dos *code smells*. Além disso, podemos testar a capacidade de generalizar os resultados dos experimentos para estudos com linguagens de programação orientadas a objetos diferentes de Java. E também desenvolver uma ferramenta para editores de código e/ou IDEs com intuito de identificação de *code smells* durante o processo de desenvolvimento do software.

Referências

Abdou, A. and Ramadan, N. (2022). Selected code-quality characteristics and metrics for internet of things systems. *Journal of Software: Evolution and Process*, 34:18.

Al Shalabi, L., Shaaban, Z., and Kasasbeh, B. (2006). Data mining: A preprocessing engine. *Journal of Computer Science*, 2(9):735–739.

Arlot, S. and Celisse, A. (2010). A survey of cross-validation procedures for model selection.

Ashraf Abdou, N. R. (2022). Severity classification of software code smells using machine learning techniques: A comparative study. *Journal of Software: Evolution and Process*, page 37.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Brown, W. H., Malveau, R. C., McCormick, H. W. S., and Mowbray, T. J. (1998). *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, Inc., USA, 1st edition.

Catal, C. (2012). Performance evaluation metrics for software fault prediction studies. *Acta Polytechnica Hungarica*, 9(4):193–206.

Ciupke, O. (1999). Automatic detection of design problems in object-oriented reengineering. In *Proceedings of technology of object-oriented languages and systems-TOOLS 30 (Cat. No. PR00278)*, pages 18–32. IEEE.

Dewangan, S., Rao, R. S., Mishra, A., and Gupta, M. (2021). A novel approach for code smell detection: An empirical study. *IEEE Access*, 9:162869–162883.

Di Nucci, D., Palomba, F., Tamburri, D. A., Serebrenik, A., and De Lucia, A. (2018). Detecting code smells using machine learning techniques: are we there yet? In *2018 ieee 25th international conference on software analysis, evolution and reengineering (saner)*, pages 612–621. IEEE.

Fontana, F. A., Ferme, V., Zanoni, M., and Roveda, R. (2015). Towards a prioritization of code debt: A code smell intensity index. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pages 16–24. IEEE.

Fontana, F. A., Mäntylä, M., Zanoni, M., and Marino, A. (2016). Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, 21:1143–1191.

Fontana, F. A. and Zanoni, M. (2017). Code smell severity classification using machine learning techniques. *Knowledge-Based Systems*, 128:43–58.

Fowler, M. (2018). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 2nd edition.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

Han, J., Pei, J., and Tong, H. (2022). *Data mining: concepts and techniques*. Morgan kaufmann.

Hossin, M. and Sulaiman, M. N. (2015). A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1.

Kaur, I. and Kaur, A. (2021). A novel four-way approach designed with ensemble feature selection for code smell detection. *IEEE Access*, 9:8695–8707.

Lanza, M. and Marinescu, R. (2007). *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media.

McConnell, S. (2004). *Code complete : a practical handbook of sofware construction*. Microsoft Press.

Mhawish, M. Y. and Gupta, M. (2020). Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics. *Journal of Computer Science and Technology*, 35:1428–1445.

Moha, N., Guéhéneuc, Y.-G., Duchien, L., and Le Meur, A.-F. (2009). Decor: A method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering*, 36(1):20–36.

Noble, W. S. (2006). What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567.

Palomba, F., Di Nucci, D., Tufano, M., Bavota, G., Oliveto, R., Poshyvanyk, D., and De Lucia, A. (2015). Landfill: An open dataset of code smells with public evaluation. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 482–485. IEEE.

Pushpalatha, M. and Mrunalini, M. (2021). Predicting the severity of open source bug reports using unsupervised and supervised techniques. In *Research Anthology on Usage and Development of Open Source Software*, pages 676–692. IGI Global.

Quinlan, J. R. (1990). Decision trees and decision-making. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):339–346.

Ray, S. (2019). A quick review of machine learning algorithms. In *2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon)*, pages 35–39. IEEE.

Riel, A. J. (1996). *Object-Oriented Design Heuristics*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition.

Ruck, D. W., Rogers, S. K., and Kabrisky, M. (1990). Feature selection using a multilayer perceptron. *Journal of neural network computing*, 2(2):40–48.

Sahin, D., Kessentini, M., Bechikh, S., and Deb, K. (2014). Code-smell detection as a bilevel problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(1):1–44.

Tarwani, S. and Chug, A. (2016). Predicting maintainability of open source software using gene expression programming and bad smells. In *2016 5th international conference on reliability, Infocom technologies and optimization (trends and future directions)(ICRITO)*, pages 452–459. IEEE.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. (2010). The qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference*, pages 336–345.

Travassos, G., Shull, F., Fredericks, M., and Basili, V. R. (1999). Detecting defects in object-oriented designs: using reading techniques to increase software quality. *ACM sigplan notices*, 34(10):47–56.

Tsantalis, N. and Chatzigeorgiou, A. (2009). Identification of move method refactoring opportunities. *IEEE Transactions on Software Engineering*, 35(3):347–367.

Tsantalis, N. and Chatzigeorgiou, A. (2011). Ranking refactoring suggestions based on historical volatility. In *2011 15th European conference on software maintenance and reengineering*, pages 25–34. IEEE.

Yamashita, A. and Counsell, S. (2013). Code smells as system-level indicators of maintainability: An empirical study. *Journal of Systems and Software*, 86(10):2639–2653.

Yamashita, A. and Moonen, L. (2012). Do code smells reflect important maintainability aspects? In *2012 28th IEEE international conference on software maintenance (ICSM)*, pages 306–315. IEEE.

Yamashita, A. and Moonen, L. (2013). Exploring the impact of inter-smell relations on software maintainability: An empirical study. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 682–691. IEEE.